Control de la configuración

Trabajo Práctico 1 - Git Básico

1- Objetivos de Aprendizaje

- Utilizar herramientas de control de configuración de software
- Familiarizarse con los comandos más utilizados
- Configurar el repositorio principal de cada alumno para la materia

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 4

3- Consignas a desarrollar en el trabajo práctico:

- Los ejercicios representan casos concretos y rutinarios en uso de este tipo de herramientas
- En los puntos donde corresponda, proveer los comandos de git necesarios para llevar a cabo el punto.
- Cuando se especifique alguna descripción, realizarlo de la manera más clara posible y con ejemplos cuando sea necesario.

4- Desarrollo:

1- Instalar Git

Los pasos y referencias asumen el uso del sistema operativo Windows, en caso otros SO seguir recomendaciones específicas.

- Bajar e instalar el cliente git. Por ejemplo, desde https://git-scm.com/
- Bajar e instalar un cliente visual. Por ejemplo, TortoiseGit para Windows o SourceTree para Windows/MAC:
  - https://tortoisegit.org/
  - https://www.sourcetreeapp.com/
  - Lista completa: https://git-scm.com/downloads/guis/

2- Crear un repositorio local y agregar archivos

- Crear un repositorio local en un nuevo directorio.
- Agregar un archivo Readme.md, agregar algunas líneas con texto a dicho archivo.
- Crear un commit y proveer un mensaje descriptivo.

3- Crear un repositorio remoto

- Crear una cuenta en https://github.com
- Crear un nuevo repositorio en dicha página (vacío)

- Asociar el repositorio local creado en el punto 2 al creado en github.
- Subir los cambios locales a github.

4- Familiarizarse con el concepto de Pull Request

Para algunos de los puntos proveer imágenes.

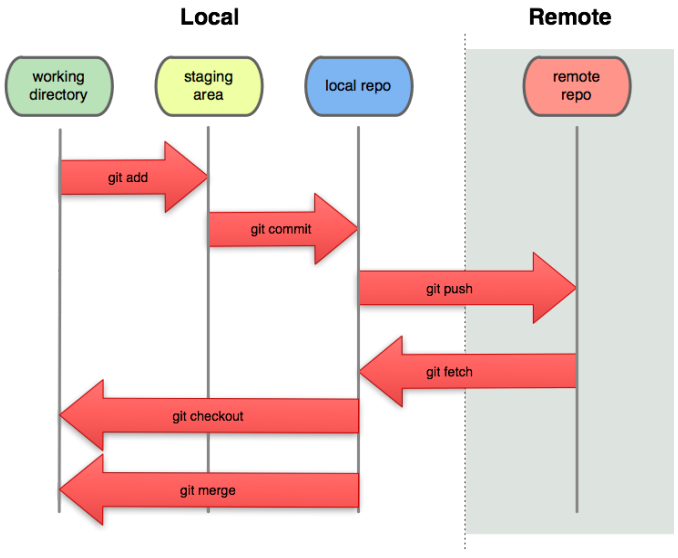
- Explicar que es un pull request.
- Crear un branch local y agregar cambios a dicho branch.
- Subir el cambio a dicho branch y crear un pull request.
- Completar el proceso de revisión en github y mergear el PR al branch master.

5- Mergear código con conflictos

- Instalar alguna herramienta de comparación. Idealmente una 3-Way:
  - P4Merge https://www.perforce.com/downloads/visual-merge-tool. Puede requerir registración
  - ByondCompare trial version https://www.scootersoftware.com/download.php
- Configurar Tortoise/SourceTree para soportar esta herramienta.
  - https://www.scootersoftware.com/support.php?zz=kb_vcs
  - http://hotkoehls.com/2015/04/use-perforce-p4merge-with-tortoisegit/
- Clonar en un segundo directorio el repositorio creado en github.
- En el clon inicial, modificar el Readme.md agregando más texto.
- Hacer commit y subir el cambio a master a github.
- En el segundo clon también agregar texto, en las mismas líneas que se modificaron el punto anterior.
- Intentar subir el cambio, haciendo un commit y push. Mostrar el error que se obtiene.
- Hacer pull y mergear el código (solo texto por ahora), mostrar la herramienta de mergeo como luce.
- Resolver los conflictos del código.
- Explicar las versiones LOCAL, BASE y REMOTE.
- Pushear el cambio mergeado.

6- Crear Repositorio de la materia

- Crear un repositorio para la materia en github. Por ejemplo ing-software-3
- Subir archivo(s) .md con los resultados e imágenes de este trabajo práctico. Puede ser en una subcarpeta trabajo-practico-01



Herramientas de construcción de software

Trabajo Práctico 2 - H. de C. de SW

1- Objetivos de Aprendizaje

- Utilizar herramientas de construcción de software y manejo de paquetes y dependencias
- Familiarizarse con las herramientas más utilizadas en el lenguaje Java.

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 4 (Libro Continuous Delivery: Cap 6 y 13)

3- Consignas a desarrollar en el trabajo práctico:

- Las aplicaciones utilizadas son del tipo "Hello World", dado que el foco del trabajo práctico es cómo construirlas y no el funcionamiento de la aplicación en sí.
- La mayoría de los ejercicios se realizarán en clase con asistencia del Jefe de trabajos prácticos.
- En los puntos en los que se pida alguna descripción, realizarlo de la manera más clara posible.

4- Desarrollo:

1- Instalar Java JDK si no dispone del mismo.

- Java 8 es suficiente
- Utilizar el instalador que corresponda a su sistema operativo
- http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- Agregar la variable de entorno JAVA_HOME
  - En Windows temporalmente se puede configurar
- `SET JAVA_HOME=C:\Program Files\Java\jdk1.8.0_181`
  - O permanentemente entrando a Variables de Entorno (Winkey + Pausa -> Opciones Avanzadas de Sistema -> Variables de Entorno)
- Otros sistemas operativos:
  - https://www3.ntu.edu.sg/home/ehchua/programming/howto/JDK_Howto.html
  - https://www.digitalocean.com/community/tutorials/how-to-install-java-with-apt-on-ubuntu-18-04

2- Instalar Ant

- Descargar Ant (archivo apache-ant-1.10.5-bin.zip) desde https://ant.apache.org/bindownload.cgi
- Descomprimir en una carpeta, por ejemplo C:\tools

- Agregar el siguiente directorio a la variable de entorno PATH, asumiendo que los binarios de Ant están en C:\tools\apache-ant-1.10.5\bin

```
SET PATH=%PATH%;C:\tools\apache-ant-1.10.5\bin
```

- Se puede modificar permanentemente la variable de sistema PATH entrando a (Winkey + Pausa -> Opciones Avanzadas de Sistema -> Variables de Entorno)
- En Linux/Mac se puede agregar la siguiente entrada a ~/.bash_profile

```
export PATH=/opt/apache-ant-1.10.5/bin:$PATH
```

3- Introducción a Ant

- Explicar brevemente qué es Ant:

Apache Ant es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build). Es, por tanto, un software para procesos de automatización de compilación, similar a Make pero desarrollado en lenguaje Java y requiere la plataforma Java, así que es más apropiado para la construcción de proyectos Java. Esta herramienta, hecha en el lenguaje de programación Java, tiene la ventaja de no depender de las órdenes del shell de cada sistema operativo, sino que se basa en archivos de configuración XML y clases Java para la realización de las distintas tareas, siendo idónea como solución multi-plataforma. La diferencia más notable entre Ant y Make es que Ant utiliza XML para describir el proceso de generación y sus dependencias, mientras que Make utiliza formato makefile. Por defecto, el archivo XML se denomina *build.xml*. Ant es un proyecto de la Apache Software Foundation. Es software open source, y se lanza bajo la licencia Apache Software.

- Analizar el contenido del ./trabajo-practico-02/ant/build.xml
- Ejecutar los siguientes comandos desde la consola:

```
ant all
ant javadoc
```

- Comentar que se obtiene con los mismos.
- Agregar al build.xml la posibilidad de generar archivo zip con la salida del proceso de build. En el target dist y usando en AntTask Zip.
- Sacar conclusiones.

3- Instalar Maven

- Instalar maven desde https://maven.apache.org/download.cgi (última versión disponible 3.4.5)
- Descomprimir en una carpeta, por ejemplo C:\tools
- Agregar el siguiente directorio a la variable de entorno PATH, asumiendo que los binarios de ant estan en C:\tools\apache-maven-3.5.4\bin
- `SET PATH=%PATH%;C:\tools\apache-maven-3.5.4\bin`
- Se puede modificar permanentemente la variable PATH entrando a (Winkey + Pausa -> Opciones Avanzadas de Sistema -> Variables de Entorno)
- En Linux/Mac se puede agregar la siguiente entrada al archivo ~/.bash_profile
- `export PATH=/opt/apache-maven-3.5.4/bin:$PATH`

4- Introducción a Maven

- Qué es Maven?

Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant (y en menor medida a PEAR de PHP y CPAN de Perl), pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero ahora ya es un proyecto de nivel superior de la Apache Software Foundation.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores. Este repositorio y su sucesor reorganizado, el repositorio Maven 2, pugnan por ser el mecanismo de facto de distribución de aplicaciones en Java, pero su adopción ha sido muy lenta. Maven provee soporte no solo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local.

Maven está construido usando una arquitectura basada en plugins que permite que utilice cualquier aplicación controlable a través de la entrada estándar. En teoría, esto podría permitir a cualquiera escribir plugins para su interfaz con herramientas como compiladores, herramientas de pruebas unitarias, etcétera, para cualquier otro lenguaje. En realidad, el soporte y uso de lenguajes distintos de Java es mínimo. Actualmente existe un plugin para .Net Framework y es mantenido, y un plugin nativo para C/C++ fue alguna vez mantenido por Maven 1.

- Qué es el archivo POM? (*En la definición de Maven*)
    - modelVersion

    Declares to which version of project descriptor this POM conforms.

    - groupId

A universally unique identifier for a project. It is normal to use a fully-qualified package name to distinguish it from other projects with a similar name (eg. org.apache.maven).

    - artifactId

The identifier for this artifact that is unique within the group given by the group ID. An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs.

    - versionId

    The current version of the artifact produced by this project.

- Repositorios Local, Central y Remotos
  http://maven.apache.org/guides/introduction/introduction-to-repositories.html
- Entender Ciclos de vida de build
    - default
    - clean
    - site
    - Referencia:
      http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Build_Lifecycle_Basics

- Comprender las fases de un ciclo de vida, por ejemplo, default:

Fase de build	Descripción
validate	validate the project is correct and all necessary information is available
compile	compile the source code of the project
test	test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
package	take the compiled code and package it in its distributable format, such as a JAR.
verify	run any checks on results of integration tests to ensure quality criteria are met
install	install the package into the local repository, for use as a dependency in other projects locally
deploy	done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

- Copiar el siguiente contenido a un archivo, por ejemplo ./trabajo-practico-02/maven/vacio/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>ar.edu.ucc</groupId>
    <artifactId>proyecto-01</artifactId>
    <version>0.1-SNAPSHOT</version>
</project>
```

- Ejecutar el siguiente comando en el directorio donde se encuentra el archivo pom.xml

`mvn clean install`

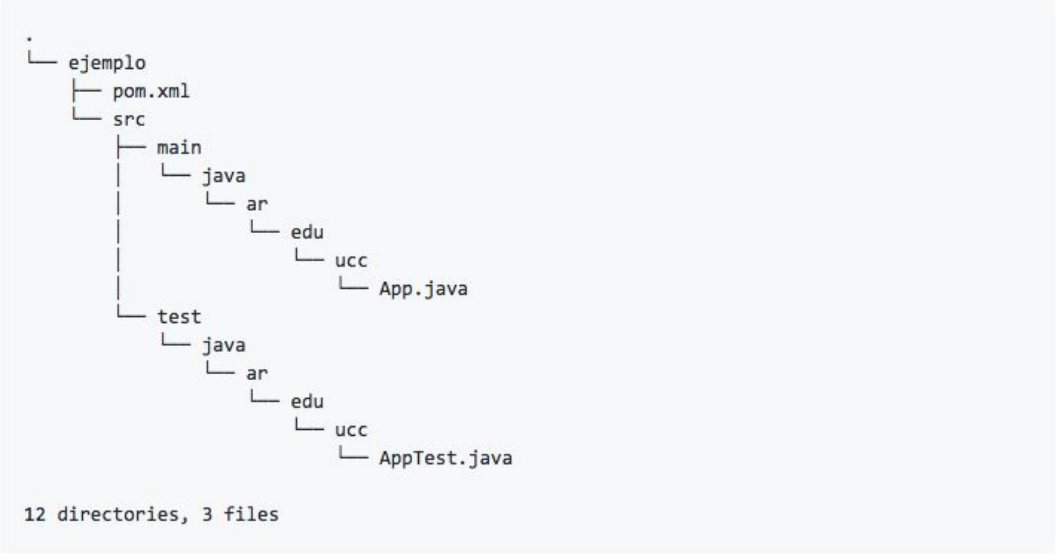
- Sacar conclusiones del resultado

5- Maven Continuación

- Generar un proyecto con una estructura inicial:

`mvn archetype:generate -DgroupId=ar.edu.ucc -DartifactId=ejemplo -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false`

- Analizar la estructura de directorios generada:



- Compilar el proyecto

`mvn clean package`

- Analizar la salida del comando anterior y luego ejecutar el programa

`java -cp target/ejemplo-1.0-SNAPSHOT.jar ar.edu.ucc.App`

6- Manejo de dependencias

- Crear un nuevo proyecto con artifactId ejemplo-uber-jar
- Modificar el código de App.java para agregar utilizar una librería de logging:

```java
package ar.edu.ucc;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        Logger log = LoggerFactory.getLogger(App.class);
        log.info("Hola Mundo!");
    }
}
```

- Compilar el código e identificar el problema.
- Agregar la dependencia necesaria al pom.xml

```xml
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.1</version>
</dependency>
```

- Verificar si se genera el archivo jar y ejecutarlo

```
java -cp target\ejemplo-uber-jar-1.0-SNAPSHOT.jar ar.edu.ucc.App
```

- Sacar conclusiones y analizar posibles soluciones
- Implementar la opción de uber-jar: https://maven.apache.org/plugins/maven-shade-plugin/

```xml
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>2.0</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
          <configuration>
            <finalName>${project.artifactId}</finalName>
            <transformers>
              <transformer implementation="org.apache.maven.plugins.shade.resource.Manifest
                <mainClass>ar.edu.ucc.App</mainClass>
              </transformer>
            </transformers>
            <minimizeJar>false</minimizeJar>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

- Volver a generar la salida y probar ejecutando

```
java -jar target\ejemplo-uber-jar.jar
```

7- Ejemplo con nodejs (Opcional, pero recomendando)

- Instalar Nodejs: https://nodejs.org/en/
- Instalar el componente para generar aplicaciones Express

`npm install express-generator -g`

- Crear una nueva aplicación

`express --view=ejs hola-mundo`

- Ejecutar la aplicación

```
cd hola-mundo
npm install
npm start
```

- La aplicación web estará disponible en http://localhost:3000
- Analizar el manejo de paquetes y dependencias realizado por npm.

8- Presentación

- Subir todo el código, ejemplos y respuestas a una carpeta trabajo-practico-02.

Tip: Agregar un archivo .gitignore al repositorio para evitar que se agreguen archivos que son resultado de la compilación u otros binarios, que no son necesarios, al mismo.

Trabajo Práctico 3 Introducción a Docker

1- Objetivos de Aprendizaje

- Familiarizarse con la tecnología de contendores
- Ejercitar comandos básicos de Docker.

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 9 (Libro Building Microservices: Cap 6)

3- Consignas a desarrollar en el trabajo práctico:

A continuación, se presentarán algunos conceptos generales de la tecnología de contenedores a manera de introducción al tema desde el punto de vista práctico.

Qué son los contenedores?

Los contenedores son paquetes de software. Ellos contienen la aplicación a ejecutar junto con las librerías, archivos de configuración, etc para que esta aplicación pueda ser ejecutada. Estos contenedores utilizan características del sistema operativo, por ejemplo, cgroups, namespaces y otros aislamientos de recursos (sistema de archivos, red, etc) para proveer un entorno aislado de ejecución de dicha aplicación.

Dado que ellos utilizan el kernel del sistema operativo en el que se ejecutan, no tienen el elevado consumo de recursos que por ejemplo tienen las máquinas virtuales, las cuales corren su propio sistema operativo.

Que es docker?

Docker es una herramienta que permite el despliegue de aplicaciones en contenedores. Además, provee una solución integrada tanto para la ejecución como para la creación de contenedores entre otras muchas funcionalidades.

Porque usar contenedores?

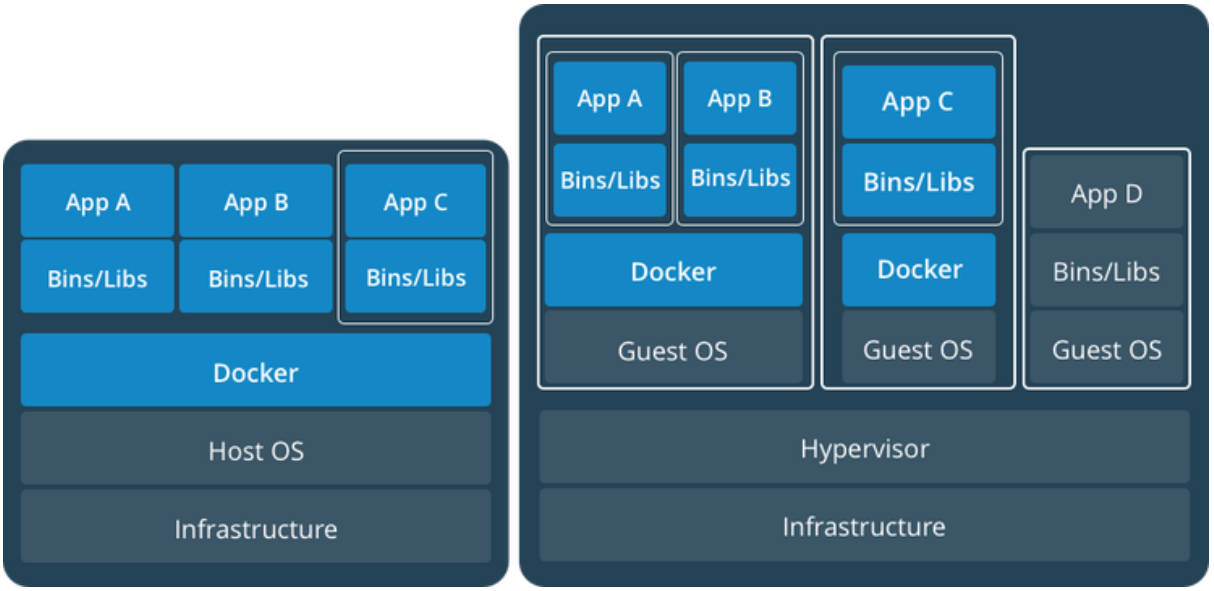
Los contenedores ofrecen un mecanismo de empaquetado lógico en el cual las aplicaciones pueden estar aisladas del entorno en el cual efectivamente se ejecutan. Este desacoplamiento permite a las aplicaciones en contenedores ser desplegadas de manera simple y consistente independientemente de si se trata de un Data Center privado, una Cloud pública, o una computadora de uso personal. Esto permite a los desarrolladores crear entornos predecibles que están aislados del resto de las aplicaciones y pueden ser ejecutados en cualquier lugar.

Por otro lado, ofrecen un control más fino de los recursos y son más eficientes al momento de la ejecución que una máquina virtual.

En los últimos años el uso de contenedores ha crecido exponencialmente y fue adoptado de forma masiva por prácticamente todas las compañías importantes de software.

Máquinas Virtuales vs Contenedores

Los contenedores no fueron pensados como un reemplazo de las máquinas virtuales. Cuando ambas tecnologías se utilizan en forma conjunta se obtienen los mejores resultados, por ejemplo, en los proveedores cloud como AWS, Google Cloud o Microsoft Azure.



Conceptos Generales

- Container Image: Una imagen contiene el sistema operativo base, la aplicación y todas sus dependencias necesarias para un despliegue rápido del contenedor.
- Container: Es una instancia en ejecución de una imagen.
- Container Registry: Las imágenes de Docker son almacenadas en un Registry y pueden ser descargadas cuando se necesitan. Un registry puede ser público, por ejemplo, DockerHub o instalado en un entorno privado.
- Docker Daemon: el servicio en segundo plano que se ejecuta en el host que gestiona la construcción, ejecución y distribución de contenedores Docker. El daemon es el proceso que se ejecuta en el sistema operativo con el que los clientes hablan.
- Docker Client: la herramienta de línea de comandos que permite al usuario interactuar con el daemon. En términos más generales, también puede haber otras formas de clientes, como Kitematic, que proporciona una GUI a los usuarios.
- Dockerfile: Son usados por los desarrolladores para automatizar la creación de imágenes de contenedores. Con un Dockerfile, el demonio de Docker puede automáticamente construir una imagen.

4- Desarrollo:

1- Instalar Docker Community Edition

- Diferentes opciones para cada sistema operativo
- https://docs.docker.com/
- Ejecutar el siguiente comando para comprobar versiones de cliente y demonio.

```
docker version
```

2- Explorar DockerHub

- Registrase en docker hub: https://hub.docker.com/
- Familiarizarse con el portal

3- Obtener la imagen BusyBox

- Ejecutar el siguiente comando, para bajar una imagen de DockerHub

```
docker pull busybox
```

- Verificar qué versión y tamaño tiene la imagen bajada, obtener una lista de imagenes locales:

```
docker images
```

4- Ejecutando contenedores

- Ejecutar un contenedor utilizando el comando run de docker:

```
docker run busybox
```

- Explicar porque no se obtuvo ningún resultado
- Especificamos algún comando a correr dentro del contendor, ejecutar por ejemplo:

```
docker run busybox echo "Hola Mundo"
```

- Ver los contendores ejecutados utilizando el comando ps:

```
docker ps
```

- Vemos que no existe nada en ejecución, correr entonces:

```
docker ps -a
```

- Mostrar el resultado y explicar que se obtuvo como salida del comando anterior.

5- Ejecutando en modo interactivo

- Ejecutar el siguiente comando

```
docker run -it busybox sh
```

- Para cada uno de los siguientes comandos dentro de contenedor, mostrar los resultados:

```
ps
uptime
free
ls -l /
```

- Salimos del contendor con:

```
exit
```

6- Borrando contendores terminados

- Obtener la lista de contendores

```
docker ps -a
```

- Para borrar podemos utilizar el id o el nombre (autogenerado si no se especifica) de contendor que se desee, por ejemplo:

```
docker rm elated_lalande
```

- Para borrar todos los contendores que no estén corriendo, ejecutar cualquiera de los siguientes comandos:

```
docker rm $(docker ps -a -q -f status=exited)
```

```
docker container prune
```

7- Montando volúmenes

Hasta este punto los contenedores ejecutados no tenían contacto con el exterior, ellos corrían en su propio entorno hasta que terminaran su ejecución. Ahora veremos como montar un volumen dentro del contenedor para visualizar por ejemplo archivos del sistema huesped:

- Ejecutar el siguiente comando, cambiar myusuario por el usuario que corresponda. En linux/Mac puede utilizarse /home/miusuario):

```
docker run -it -v C:\Users\misuario\Desktop:/var/escritorio busybox /bin/sh
```

- Dentro del contenedor correr

```
ls -l /var/escritorio
touch /var/escritorio/hola.txt
```

- Verificar que el Archivo se ha creado en el escritorio o en el directorio home según corresponda.

8- Publicando puertos

En el caso de aplicaciones web o base de datos donde se interactúa con estas aplicaciones a través de un puerto al cual hay que acceder, estos puertos están visibles solo dentro del contenedor. Si queremos acceder desde el exterior deberemos exponerlos.

- Ejecutar la siguiente imagen, en este caso utilizamos la bandera -d (detach) para que nos devuelva el control de la consola:

```
docker run -d daviey/nyan-cat-web
```

- Si ejecutamos un comando ps:

```
PS D:\> docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS          PORTS                NAMES
87d1c5f44809   daviey/nyan-cat-web  "nginx -g 'daemon of..."  2 minutes ago    Up 2 minutes    80/tcp, 443/tcp      compassionate_raman
```

- Vemos que el contendor expone 2 puertos el 80 y el 443, pero si intentamos en un navegador acceder a http://localhost no sucede nada.
- Procedemos entonces a parar y remover este contenedor:

```
docker kill compassionate_raman
docker rm compassionate_raman
```

- Vamos a volver a correrlo otra vez, pero publicando uno de los puertos solamente, el este caso el 80

```
docker run -d -p 80:80 daviey/nyan-cat-web
```

- Accedamos nuevamente a http://localhost y expliquemos que sucede.

8- Presentación del trabajo práctico.

Subir un archivo trabajo-practico-03.md con las salidas de los comandos utilizados. Si es necesario incluir también capturas de pantalla.

Trabajo Práctico 4 Imágenes de Docker

1- Objetivos de Aprendizaje

- Adquirir conocimientos para construir y publicar imágenes de Docker.
- Familiarizarse con el vocabulario.

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 4

3- Consignas a desarrollar en el trabajo práctico:

- Los ejercicios se realizarán en clase con asistencia del Jefe de trabajos prácticos.
- En los puntos en los que se pida alguna descripción, realizarlo de la manera más clara posible.

4- Desarrollo:

1- Conceptos de Dockerfiles

- Leer https://docs.docker.com/engine/reference/builder/ (tiempo estimado 2 horas)
- Describir las instrucciones
  - FROM
  - RUN
  - ADD
  - COPY
  - EXPOSE
  - CMD
  - ENTRYPOINT

2- Generar imagen de docker

- Utilizar el ejercicio del trabajo práctico 2 ejemplo-uber-jar
- Copiar el código (pom.xml y carpeta src) a una carpeta trabajo-practico-04/java-dockeren nuestro repositorio
- Compilar la salida con:

```
mvn clean package
```

- Agregar el siguiente Dockerfile

```
FROM openjdk:8u171-jre-alpine

RUN apk add --no-cache bash

WORKDIR /opt

COPY target/ejemplo.jar .

ENV JAVA_OPTS="-Xms32m -Xmx128m"

ENTRYPOINT exec java $JAVA_OPTS -jar ejemplo.jar
```

- Generar la imagen de docker con el comando build

```
docker build -t test-java .
```

- Ejecutar el contenedor

```
docker run test-java
```

- Capturar y mostrar la salida.

3- Imagen para aplicación web en Nodejs

- Actualizar el repositorio de la materia
- Abrir la carpeta trabajo-practico-04/nodejs-docker
- Escribir un Dockerfile para ejecutar la aplicación web localizada en ese directorio
    - Usar como imagen base node:8.11-alpine
    - Ejecutar npm install dentro durante el build.
    - Exponer el puerto 3000
- Hacer un build de la imagen, nombrar la imagen test-node.
- Ejecutar la imange test-node publicando el puerto 3000.
- Verificar en http://localhost:3000 que la aplicación está funcionando.
- Proveer el Dockerfile y los comandos ejecutados como resultado de este ejercicio.

4- Publicar la imagen en Docker Hub.

- Crear una cuenta en Docker Hub si no se dispone de una.
- Registrase localmente a la cuenta de Docker Hub:

```
docker login
```

- Crear un tag de la imagen generada en el ejercicio 3. Reemplazar <mi_usuario> por el creado en el punto anterior.

```
docker tag test-node <mi_usuario>/test-node:latest
```

- Subir la imagen a Docker Hub con el comando

```
docker push <mi_usuario>/test-node:latest
```

- Como resultado de este ejercicio mostrar la salida de consola, o una captura de pantalla de la imagen disponible en Docker Hub.

Integración Continua

Trabajo Práctico 5 -
Primeros pasos con Jenkins

1- Objetivos de Aprendizaje

- Adquirir conocimientos acerca de las herramientas de integración continua.
- Configurar este tipo de herramientas.
- Implementar procesos de construcción automatizado simples.

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 4 (Libro Continuous Delivery: Cap 3)

3- Consignas a desarrollar en el trabajo práctico:

- Los ejercicios se realizarán en clase con asistencia del Jefe de trabajos prácticos.
- Para una mejor evaluación del trabajo práctico, incluir capturas de pantalla de los pasos donde considere necesario.

4- Desarrollo:

1- Poniendo en funcionamiento Jenkins

- Ejecutar el contenedor de Jenkins (tamaño de imagen de docker ~700mb). Utilizar otro jenkins_home (C:\data\jenkins) si considera necesario:

```
# Windows

mkdir C:\data\jenkins
docker run -d -p 8080:8080 -p 50000:50000 -v C:\data\jenkins:/var/jenkins_home
jenkins/jenkins:lts

# Linux / Mac OS

mkdir -p ~/data/jenkins
docker run -d -p 8080:8080 -p 50000:50000 -v ~/data/jenkins:/var/jenkins_home
jenkins/jenkins:lts
```

- Una vez en ejecución, abrir http://localhost:8080
- Inicialmente deberá especificar el texto dentro del archivo C:\data\jenkins\secrets\initialAdminPassword
- Instalar los plugins por defecto
- Crear el usuario admin inicial. Colocar cualquier valor que considere adecuado.

2- Conceptos generales

- Junto al Jefe de trabajos prácticos:
    - Explicamos los diferentes componentes que vemos en la página principal
    - Analizamos las opciones de administración de Jenkins

Jenkins es un software de Integración continua open source (no confundir con Leeroy Jenkins.) escrito en Java. Está basado en el proyecto Hudson y es, dependiendo de la visión, un fork del proyecto o simplemente un cambio de nombre.

Jenkins proporciona integración continua para el desarrollo de software. Es un sistema corriendo en un servidor que es un contenedor de servlets, como Apache Tomcat. Soporta herramientas de control de versiones como CVS, Subversion, Git, Mercurial, Perforce y Clearcase y puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como scripts de shell y programas batch de Windows. El desarrollador principal es Kohsuke Kawaguchi. Liberado bajo licencia MIT, Jenkins es software libre.

3- Instalando Plugins y configurando herramientas

- En Administrar Jenkins vamos a la sección de Administrar Plugins
- De la lista de plugins disponibles instalamos Maven Integration plugin
- Instalamos sin reiniciar el servidor.
- Abrir nuevamente página de Plugins y explorar la lista, para familiarizarse qué tipo de plugins hay disponibles.
- En la sección de administración abrir la opción de configuración de herramientas
- Agregar maven con el nombre de M3 y que se instale automáticamente.

4- Creando el primer Pipeline Job

- Crear un nuevo item, del tipo Pipeline con nombre hello-world
- Una vez creado el job, en la sección Pipeline seleccionamos try sample Pipeline y luego Hello World
- Guardamos y ejecutamos el Job
- Analizar la salida del mismo

5- Creando un Pipeline Job con Git y Maven

- Similar al paso anterior creamos un ítem con el nombre simple-maven
- Elegir Git + Maven en la sección try sample Pipeline
- Guardar y ejecutar el Job
- Analizar el script, para identificar los diferentes pasos definidos y correlacionarlos con lo que se ejecuta en el Job y se visualiza en la página del Job.

6- Utilizando nuestros proyectos

- Utilizando lo aprendido en el ejercicio 5
    - Crear un Job que construya el proyecto ejemplo-uber-jar del trabajo práctico 2.
    - Obtener el código desde el repositorio de cada alumno (se puede crear un repositorio nuevo en github que contenga solamente el proyecto maven).
    - Generar y publicar los artefactos que se producen.
- Como resultado de este ejercicio proveer el script en el archivo trabajo-practico-05/Jenkinsfile

Trabajo Práctico 6 - Herramientas de construcción en la nube

1- Objetivos de Aprendizaje

- Adquirir conocimientos acerca de las herramientas de integración continua en la nube.
- Configurar este tipo de herramientas.
- Implementar procesos simples de construcción automatizada.

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 4 (Libro Continuous Delivery: Cap 3)

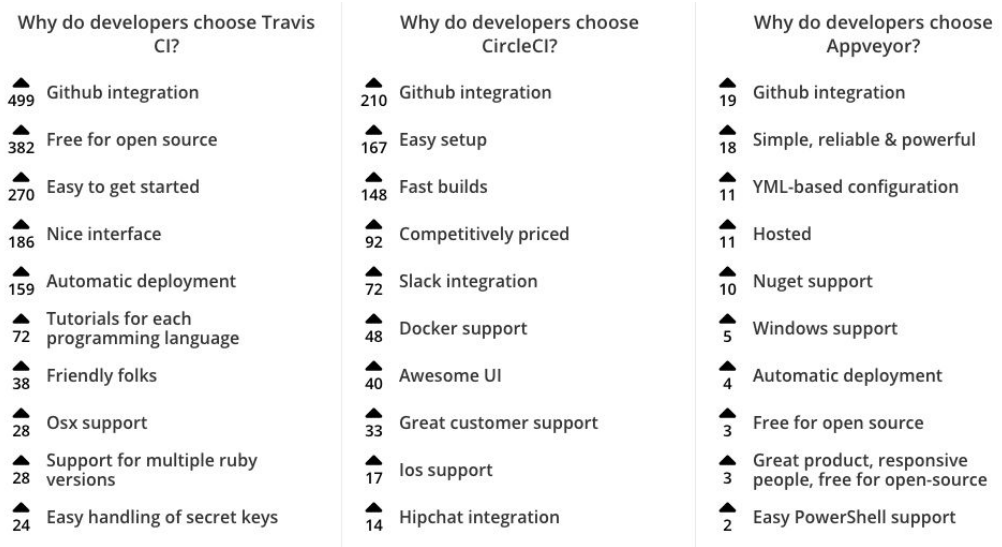
3- Consignas a desarrollar en el trabajo práctico:

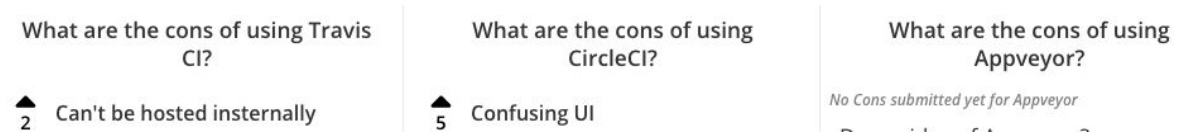
- Los ejercicios se realizarán en clase con asistencia del Jefe de trabajos prácticos.
- Para una mejor evaluación del trabajo práctico, incluir capturas de pantalla de los pasos donde considere necesario.
- En este caso existen varias herramientas con funcionalidades similares en la nube:
  - AppVeyor
  - CircleCI
  - Travis CI

4- Desarrollo:

1- Pros y Contras

- Listar los pros y contras de este tipo de herramientas
- Sacar conclusiones





2- Configurando AppVeyor

- Crear una cuenta en AppVeyor, se puede utilizar el usuario de GitHub
- Configurar un proyecto utilizando un repositorio que contenga el código del proyecto Maven ejemplo-uber-jar del trabajo práctico 2.
- Ir a la opción SETTINGS
- Dejar el entorno por defecto (Visual Studio 2015)
- En la opción Build configurar un script de línea de comando (CMD), para que genere los artefactos. Es posible que tenga que cambiar de directorio para generación:

```
cd <lugar donde está el pom.xml>
mvn clean package
```

- En la opción Artifacts especificar la ruta relativa para capturar los jar files de salida.
- Verificar que el Job se ejecuta con nuevos commits en el repositorio configurado.
- Opcional: Agregar Badges al repositorio para mostrar estado actual del build en GitHub.
- Como resultado de este ejercicio, exportar el yaml generado y subirlo entrabajo-practico-06/appveyor.yml. Y mostrar capturas de pantalla con los artefactos y/o la historia de los builds ejecutados.

3- Configurando CircleCI

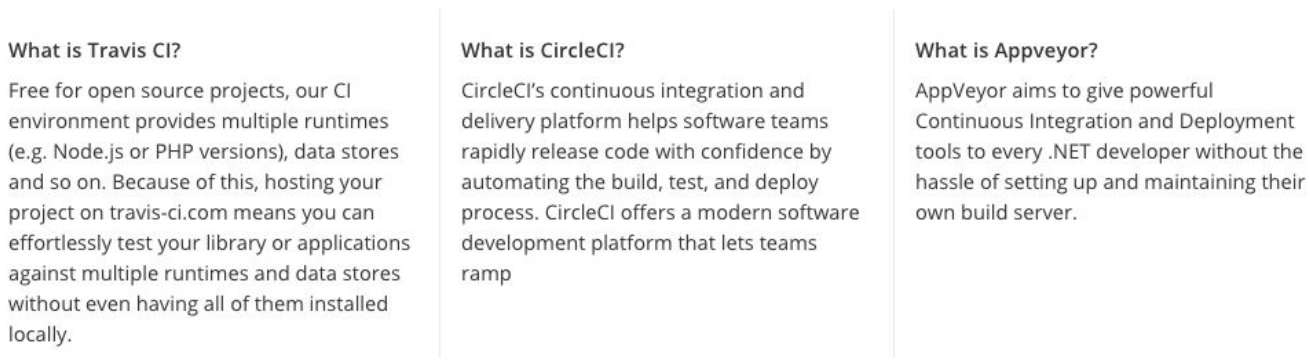
- De manera similar al ejercicio 2, configurar un build job para el mismo proyecto, pero utilizando CircleCI
- Para capturar artefactos, utilizar esta referencia: https://circleci.com/docs/2.0/artifacts/
- Como resultado de este ejercicio, subir el config.yml a la carpeta trabajo-practico-06

4- Opcional: Configurando TravisCI

- Configurar el mismo proyecto, pero para TravisCI. No es necesario publicar los artefactos porque TravisCI no dispone de esta funcionalidad.
- Como resultado de este ejercicio subir el archivo .travis.yml a la carpeta trabajo-practico-06

5- Conclusiones.

- Hacer una breve descripción comparativa entre AppVeyor, CircleCI y TravisCI



Métricas

Trabajo Práctico 7 - Métricas de código

1- Objetivos de Aprendizaje

- Adquirir conocimientos acerca del uso de herramientas para análisis estático de código.
- Configurar este tipo de herramientas.
- Prevención y corrección de errores.

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 5 (Libro Ingeniería de Software: Cap 24)

3- Consignas a desarrollar en el trabajo práctico:

- Los ejercicios se realizarán en clase con asistencia del Jefe de trabajos prácticos.
- Para una mejor evaluación del trabajo práctico, incluir capturas de pantalla de los pasos donde considere necesario.
- En este caso existen varias herramientas con funcionalidades similares en la nube:
  - SonarClould
  - Codacy

4- Desarrollo:

1- Configurando SonarCloud

- Crear una cuenta en SonarCloud, se puede utilizar el usuario de GitHub.
- Hacer un fork del repositorio https://github.com/alexisfr/java-projects
- Clonar el repositorio al que se le hizo el fork.
- En SonarCloud crear un nuevo proyecto (haciendo click en el botón +)
- Seleccionar el repositorio java-projects de su usuario y presionar Create
- Luego generar un token (puede ser cualquier palabra) y luego continuar)

2- Correr Análisis de código

- Seleccionar Java y Maven.
- Correr el comando generado en la raíz del repositorio clonado locamente, hay que agregar la opción -Dmaven.test.failure.ignore=true al final del comando, algo similar a esto:

```
mvn sonar:sonar \
  -Dsonar.projectKey=alexisfr_java-projects \
  -Dsonar.organization=alexisfr-github \
  -Dsonar.host.url=https://sonarcloud.io \
```

```
-Dsonar.login=XXX \
-Dmaven.test.failure.ignore=true
```

- Esta ejecución puede tomar varios minutos.
- Guardar este comando, que luego será utilizado.

3- Analizar los resultados de Fiabilidad

- Encontrar los 3 Bugs en la clase basics/swing/FileIO referidos a Fiabilidad (Reliability).
- Explicar porque se consideran errores y posible solución a los mismos.

4- Analizar las Vulnerabilidades de Seguridad

- Encontrar los 4 Errores de seguridad en la clase basics/jdbc/mysql/MySQLAccess.java
- Explicar porque se considera código vulnerable y posible solución a los mismos.

Las vulnerabilidades son puntos débiles de un sistema informático (compuesto por hardware, software e incluso humanos) que permiten que un atacante comprometa la integridad, disponibilidad o confidencialidad del mismo. Algunas de las vulnerabilidades más severas permiten que los atacantes ejecutar código arbitrario, denominadas vulnerabilidades de seguridad, en un sistema comprometido.

5- Analizar costo de Mantenimiento

- Describir alguno de los "Code Smells" en la clase basics/swing/demo/calculator/Calculator.java
- ¿Cuánto tiempo estimado es necesario para llevar esta clase de calificación B a calificación A de acuerdo a esta herramienta?

6- Analizar Complejidad

- Encontrar la función que posee la mayor complejidad ciclomática en la clase basics/xml/EvalXML.java
- ¿Qué significa complejidad ciclomática y complejidad cognitiva?

La **Complejidad Ciclomática** (en inglés, Cyclomatic Complexity) es una métrica del software en ingeniería del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Es una de las métricas de software de mayor aceptación, ya que ha sido concebida para ser independiente del lenguaje.

La **Complejidad Cognitiva** es una medida de cómo es de difícil entender intuitivamente un bloque de código. A diferencia de la Complejidad Ciclomática, que determina qué dificultad tiene probar el código.

- ¿Cuánto son los valores para esta clase?

7- Corregir Errores

- Encontrar el código duplicado en en la clase scheduler/TaskScheduler.java
- Refactorizar el código para eliminar este código duplicado.
- Hacer commit y push del código corregido.
- Volver a correr el análisis de código con el comando ejecutado en el ejercicio 2.
- Comprobar que la en esta clase no se reporta más dicho problema.

Automatización de Tests

Trabajo Práctico 8 Pruebas de unidad

1- Objetivos de Aprendizaje

- Adquirir conocimientos sobre conceptos referidos a pruebas de unidad (unit tests).
- Generar y ejecutar pruebas unitarias utilizado frameworks disponibles.

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 6 (Libro Ingeniería de Software: Cap 8)

3- Consignas a desarrollar en el trabajo práctico:

Conceptos generales explicaciones de los mismos

¿Qué son las pruebas de software?

Una prueba de software es una pieza de software que ejecuta otra pieza de software. Valida si ese código da como resultado el estado esperado (prueba de estado) o ejecuta la secuencia de eventos esperados (prueba de comportamiento).

¿Por qué son útiles las pruebas de software?

Las pruebas de la unidad de software ayudan al desarrollador a verificar que la lógica de una parte del programa sea correcta.

Ejecutar pruebas automáticamente ayuda a identificar regresiones de software introducidas por cambios en el código fuente. Tener una cobertura de prueba alta de su código le permite continuar desarrollando características sin tener que realizar muchas pruebas manuales.

Código (o aplicación) bajo prueba

El código que se prueba generalmente se llama código bajo prueba . Si está probando una aplicación, esto se llama la aplicación bajo prueba .

Prueba unitarias (Unit Tests)

Una prueba de unidad es una pieza de código escrita por un desarrollador que ejecuta una funcionalidad específica en el código que se probará y afirma cierto comportamiento o estado.

El porcentaje de código que se prueba mediante pruebas unitarias generalmente se llama cobertura de prueba.

Una prueba unitaria se dirige a una pequeña unidad de código, por ejemplo, un método o una clase. Las dependencias externas deben eliminarse de las pruebas unitarias, por ejemplo, reemplazando la dependencia con una implementación de prueba o un objeto (mock) creado por un framework de prueba.

Las pruebas unitarias no son adecuadas para probar la interfaz de usuario compleja o la interacción de componentes. Para esto, debes desarrollar pruebas de integración.

Frameworks de pruebas unitarias para Java

Hay varios frameworks de prueba disponibles para Java. Los más populares son JUnit y TestNG

¿Qué parte del software debería probarse?

Lo que debe probarse es un tema muy controvertido. Algunos desarrolladores creen que cada declaración en su código debe ser probada.

En cualquier caso, debe escribir pruebas de software para las partes críticas y complejas de su aplicación. Si introduce nuevas funciones, un banco de pruebas sólido también lo protege contra la regresión en el código existente.

En general, es seguro ignorar el código trivial. Por ejemplo, es inútil escribir pruebas para los métodos getter y setter que simplemente asignan valores a los campos. Escribir pruebas para estas afirmaciones consume mucho tiempo y no tiene sentido, ya que estaría probando la máquina virtual Java. La propia JVM ya tiene casos de prueba para esto. Si está desarrollando aplicaciones de usuario final, puede suponer que una asignación de campo funciona en Java.

Si comienza a desarrollar pruebas para una base de código existente sin ninguna prueba, es una buena práctica comenzar a escribir pruebas para el código en el que la mayoría de los errores ocurrieron en el pasado. De esta manera puede enfocarse en las partes críticas de su aplicación.

4- Desarrollo:

1- Familiarizarse con algunos conceptos del framework JUnit:

JUnit 4	Descripción
import org.junit.*	Instrucción de importación para usar las siguientes anotaciones.
@Test	Identifica un método como un método de prueba.

@Before	Ejecutado antes de cada prueba. Se utiliza para preparar el entorno de prueba (por ejemplo, leer datos de entrada, inicializar la clase).
@After	Ejecutado después de cada prueba. Se utiliza para limpiar el entorno de prueba (por ejemplo, eliminar datos temporales, restablecer los valores predeterminados). También puede ahorrar memoria limpiando costosas estructuras de memoria.
@BeforeClass	Ejecutado una vez, antes del comienzo de todas las pruebas. Se usa para realizar actividades intensivas de tiempo, por ejemplo, para conectarse a una base de datos. Los métodos marcados con esta anotación deben definirse static para que funcionen con JUnit.
@AfterClass	Ejecutado una vez, después de que se hayan terminado todas las pruebas. Se utiliza para realizar actividades de limpieza, por ejemplo, para desconectarse de una base de datos. Los métodos anotados con esta anotación deben definirse static para que funcionen con JUnit.
@Ignore o @Ignore("Why disabled")	Marca que la prueba debe ser deshabilitada. Esto es útil cuando se ha cambiado el código subyacente y el caso de prueba aún no se ha adaptado. O si el tiempo de ejecución de esta prueba es demasiado largo para ser incluido. Es una mejor práctica proporcionar la descripción opcional, por qué la prueba está deshabilitada.
@Test (expected = Exception.class)	Falla si el método no arroja la excepción nombrada.
@Test(timeout=100)	Falla si el método tarda más de 100 milisegundos.

Declaración	Descripción
fail ([mensaje])	Deja que el método falle. Se puede usar para verificar que no se llegue a una determinada parte del código o para realizar una prueba de falla antes de implementar el código de prueba. El parámetro del mensaje es opcional.
assertTrue ([mensaje,] condición booleana)	Comprueba que la condición booleana es verdadera.

assertFalse ([mensaje,] condición booleana)	Comprueba que la condición booleana es falsa.
assertEquals ([mensaje,] esperado, real)	Comprueba que dos valores son iguales. Nota: para las matrices, la referencia no se verifica en el contenido de las matrices.
assertEquals ([mensaje,] esperado, real, tolerancia)	Pruebe que los valores float o double coincidan. La tolerancia es el número de decimales que debe ser el mismo.
assertNull (objeto [mensaje,])	Verifica que el objeto sea nulo.
assertNotNull (objeto [mensaje,])	Verifica que el objeto no sea nulo.
assertSame ([mensaje,] esperado, real)	Comprueba que ambas variables se refieren al mismo objeto.
assertNotSame ([mensaje,] esperado, real)	Comprueba que ambas variables se refieren a diferentes objetos.

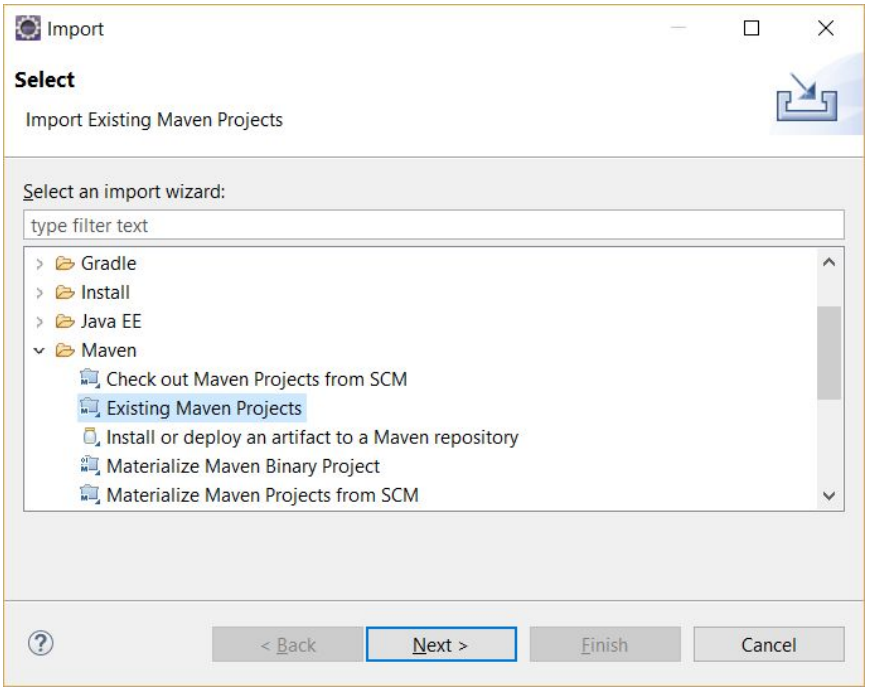
2- Configurar Proyecto para correr unit tests

- Utilizar un nuevo proyecto llamando unit-tests, para generarlo correr el siguiente comando:

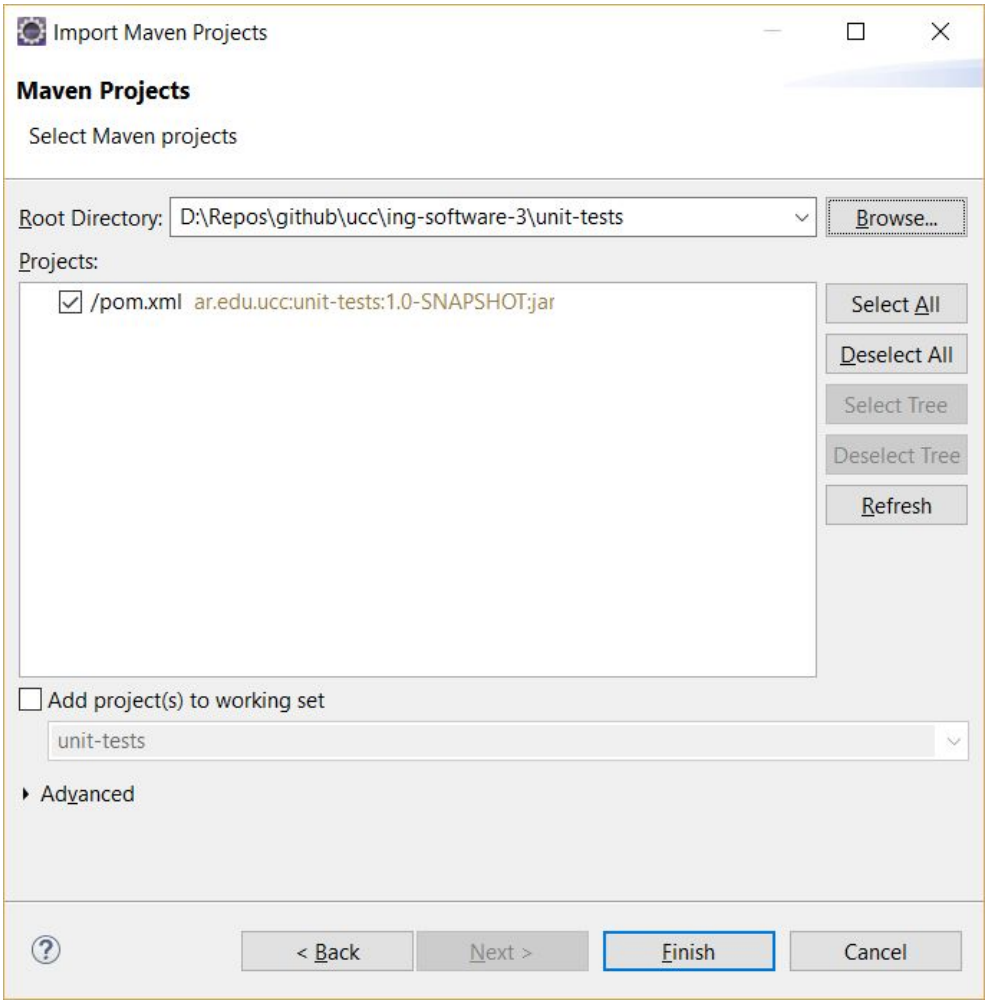
```
mvn archetype:generate -DgroupId=ar.edu.ucc -DartifactId=unit-tests
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

- Importar el proyecto en Eclipse o Intellij como maven project:
  - Si no dispone de Eclipse puede obtenerlo desde este link http://www.eclipse.org/downloads/packages/release/2018-09/r/eclipse-ide-java-ee-developers

- Para importar, ir al menú Archivo -> Importar -> Maven -> Proyecto Maven Existente:



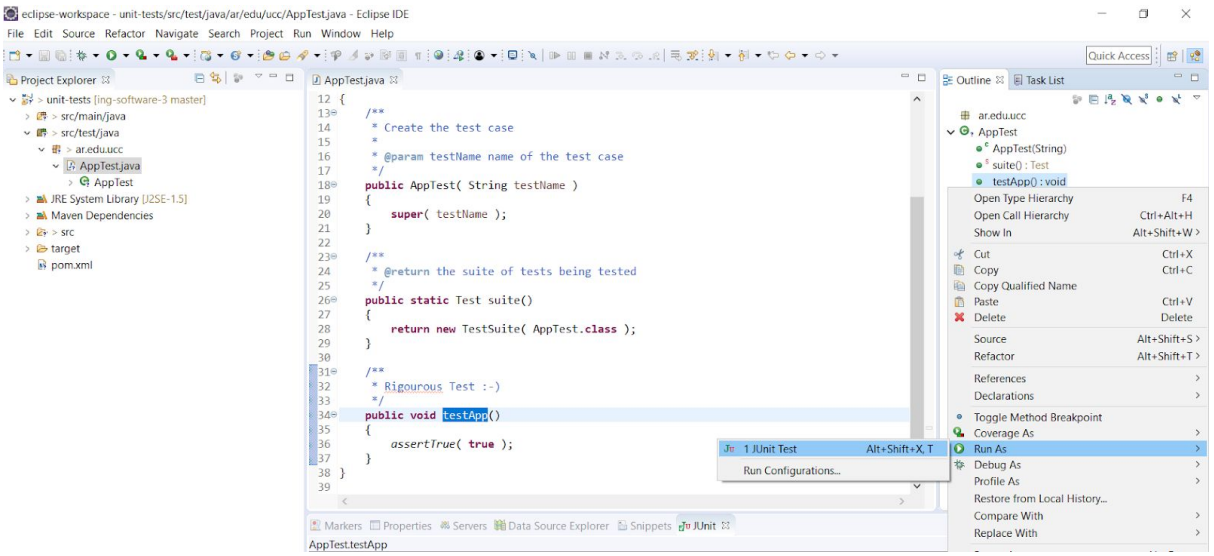
- Seleccionar el directorio donde se encuentra el pom.xml que se generó en el punto anterior. Luego continuar:



- Modificar el pom.xml para utilizar una versión más actualizada de JUnit

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

- Ejecutar el test case abriendo la clase de test src/java/test/AppTest.java. Luego en el panel Outline seleccionar el test y "ejecutar como JUnit" como se muestra en la siguiente figura:

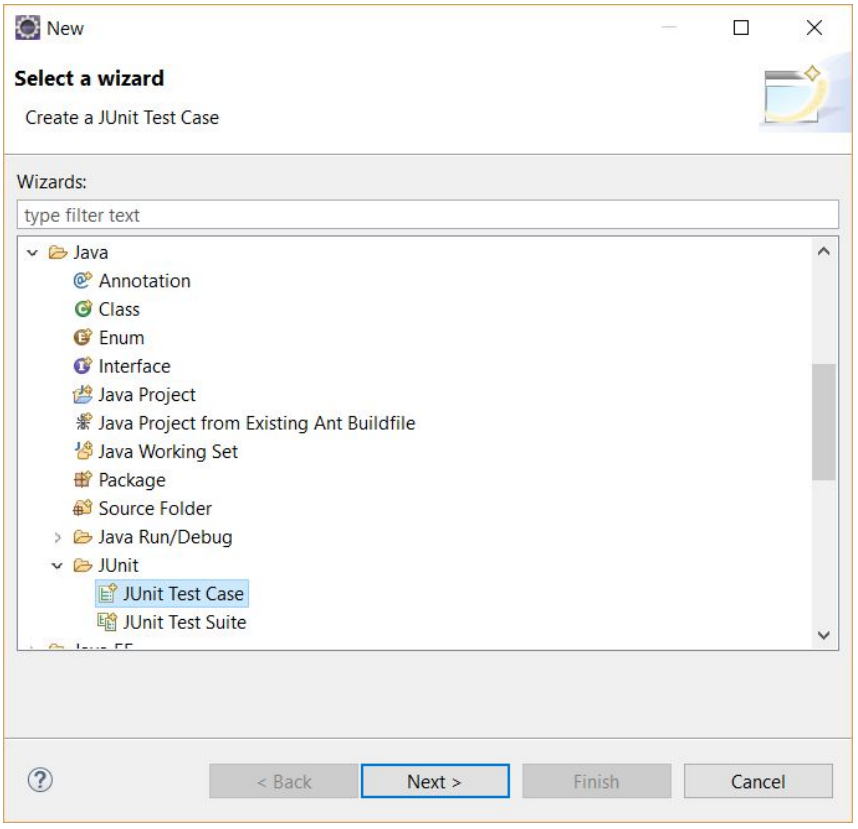


3- Escribiendo test cases para una clase

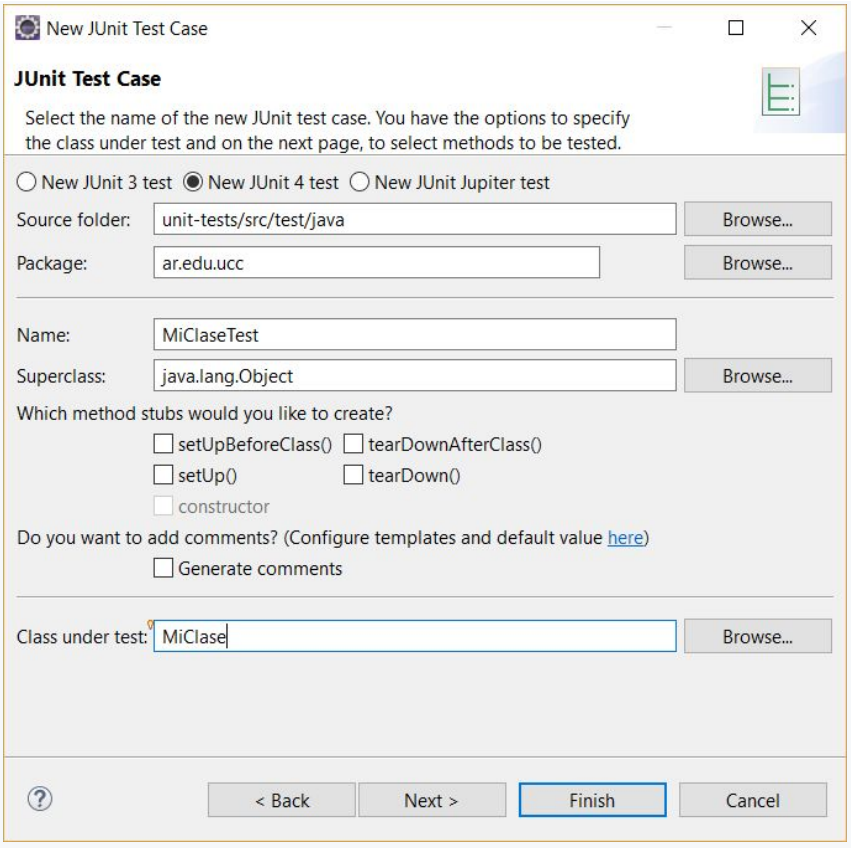
- Crear la clase MiClase en la carpeta src/main/java, en el paquete ar.edu.ucc.
  - Se puede utilizar la opción *Archivo -> Nuevo -> Clase* en Eclipse
- Modificar el código para que contenga el siguiente método:

```
package ar.edu.ucc;

public class MiClase {

	public int multiplicar(int x, int y) {
		if (x > 999) {
			throw new IllegalArgumentException("X debe ser menor a 1000");
		}
		return x / y;
	}
}
```

- Agregar un test case a este método
  - Hacer botón derecho sobre el navegador de paquetes, luego Nuevo -> Otros -> Java -> JUnit -> JUnit Test Case



- Elegir nombre del test case y clase a testear como indica la figura



- Modificar el contenido de la clase MiClaseTest para que tenga los dos siguientes métodos:

```
@Test(expected = IllegalArgumentException.class)
public void testExceptionIsThrown() {
        MiClase tester = new MiClase();
        tester.multiplicar(1000, 5);
}

@Test
public void testMultiply() {
        MiClase tester = new MiClase();
        assertEquals("10 x 5 must be 50", 50, tester.multiplicar(10, 5));
}
```

- Explicar para qué es cada uno, encontrar errores y corregir el código.

4- Familiarizarse con algunos conceptos de Mockito

Mockito es un framework de simulación popular que se puede usar junto con JUnit. Mockito permite crear y configurar objetos falsos. El uso de Mockito simplifica significativamente el desarrollo de pruebas para clases con dependencias externas.

Si se usa Mockito en las pruebas, normalmente:

1. Se burlan las dependencias externas e insertan los mocs en el código bajo prueba
2. Se ejecuta el código bajo prueba
3. Se valida que el código se ejecutó correctamente

5- Creando un Mock para cliente de Facebook

- Crear la siguiente interfaz con este contenido:

```
package ar.edu.ucc;

public interface IFacebook {
        String connect();
}
```

- Crear el siguiente clase con este código:

```
package ar.edu.ucc;

public class FacebookClient {

   public void createPost(IFacebook fb) {
      String message = fb.connect();

      System.out.println(message);

      // Otro código con la conexión ya hecha
   }
}
```

- Agregamos esta dependencia al pom.xml

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>2.7.22</version>
    <scope>test</scope>
</dependency>
```

- Luego generamos el siguiente test, como generamos el anterior, y ponemos este contenido en el archivo:

```
package ar.edu.ucc;

import org.junit.Test;
import static org.mockito.Mockito.*;

public class FacebookClientTest {

        @Test
        public void testCreatePost() {
            FacebookClient client = new FacebookClient();

            IFacebook iFacebook = mock(IFacebook.class);

            when(iFacebook.connect()).thenReturn("Using mockito is great");

            client.createPost(iFacebook);
        }
}
```

- Corremos el unit test y vemos los resultados
- Al final del unit test agregamos esta línea:

```
    verify(iFacebook, atLeastOnce()).connect();
```

- Explicar que chequea esta última sentencia en la ejecución del test case.

Resolución de práctico

- Subir todo el código del proyecto una carpeta trabajo-practico-07 en su repositorio de la materia.

Trabajo Práctico 9 Pruebas de Integración

1- Objetivos de Aprendizaje

- Adquirir conocimientos sobre conceptos referidos a pruebas de integración (integration tests).
- Generar y ejecutar pruebas de integración utilizado frameworks de código abierto.

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 6 (Libro Ingeniería de Software: Cap 8)

3- Consignas a desarrollar en el trabajo práctico:

Conceptos generales explicaciones de los mismos

Pruebas de integración

Una prueba de integración tiene como objetivo probar el comportamiento de un componente o la integración entre un conjunto de componentes. El término prueba funcional se usa a veces como sinónimo para prueba de integración. Las pruebas de integración comprueban que todo el sistema funciona según lo previsto, por lo que reducen la necesidad de pruebas manuales intensivas.

Este tipo de pruebas le permiten traducir sus historias de usuario en un conjunto de pruebas. La prueba se asemejaría a una interacción esperada del usuario con la aplicación.

Frameworks de pruebas

Existen una gran variedad de herramientas o frameworks disponibles para las pruebas de integración, tanto para componentes del backend como del frontend. Estas pueden ser comerciales, de código abierto o desarrolladas y utilizadas internamente por las compañías de software.

Para este trabajo práctico vamos a probar aplicaciones web y para ello utilizaremos Selenium como ejemplo.

Selenium

Selenium es una herramienta de prueba de software automatizada y de código abierto para probar aplicaciones web. Tiene capacidades para operar en diferentes navegadores y sistemas operativos. Selenium es un conjunto de herramientas que ayuda a los testers a automatizar las aplicaciones basadas en la web de manera más eficiente.

En este trabajo práctico utilizaremos la opción WebDriver, que nos permite enviar comandos directamente al navegador y obtener resultados. Además, podemos codificar las pruebas directamente en un lenguaje de programación, por ejemplo, Java y ejecutarlas como parte del proceso de CI/CD.

4- Desarrollo:

1- Familiarizarse con Selenium WebDriver

Comando	Descripcion
driver.get ("URL")	Navegar a una aplicación.
element.sendKeys("inputtext")	Introduce algún texto en un cuadro de entrada.
element.clear()	Borra los contenidos del cuadro de entrada.
select.deselectAll()	De-selecciona todas las OPCIONES del primer SELECCIONAR en la página.
select.selectByVisibleText("algo de texto")	Selecciona la opción con la entrada especificada por el usuario.
driver.switchTo().window("windowName")	Mueve el foco de una ventana a otra.
driver.switchTo().frame("frameName")	Cambia de marco a marco.
driver.switchTo().alert()	Ayuda en el manejo de alertas.
driver.navigate().to("URL")	Vaya a la URL.
driver.navigate().forward)	Navega hacia adelante.
driver.navigate().back()	Navegar hacia atrás.
driver.close()	Cierra el navegador actual asociado con el controlador.

Familiarizarse con los Localizadores de Selenium

La localización de elementos en Selenium WebDriver se realiza con la ayuda de los métodos findElement () y findElements () proporcionados por las clases WebDriver y WebElement.

- findElement () devuelve un objeto WebElement basado en un criterio de búsqueda específico o termina lanzando una excepción si no encuentra ningún elemento que coincida con el criterio de búsqueda.

- findElements () devuelve una lista de WebElements que coinciden con los criterios de búsqueda. Si no se encuentran elementos, devuelve una lista vacía.

Método	Sintaxis	Descripción
Por identificación	driver.findElement(By.id ())	Localiza un elemento usando el atributo ID
Por nombre	driver.findElement(By.name ())	Localiza un elemento usando el atributo Nombre
Por nombre de clase	driver.findElement(By.className ())	Localiza un elemento usando el atributo Clase
Por nombre de etiqueta	driver.findElement(By.tagName ())	Localiza un elemento usando la etiqueta HTML
Por enlace de texto	driver.findElement(By.linkText ())	Localiza un enlace usando el texto del enlace.
Por texto de enlace parcial	driver.findElement(By.partialLinkText ())	Localiza un enlace usando el texto parcial del enlace.
Por CSS	driver.findElement(By.cssSelector ())	Localiza un elemento utilizando el selector CSS.
Por XPath	driver.findElement(By.xpath ())	Localiza un elemento usando la consulta XPath

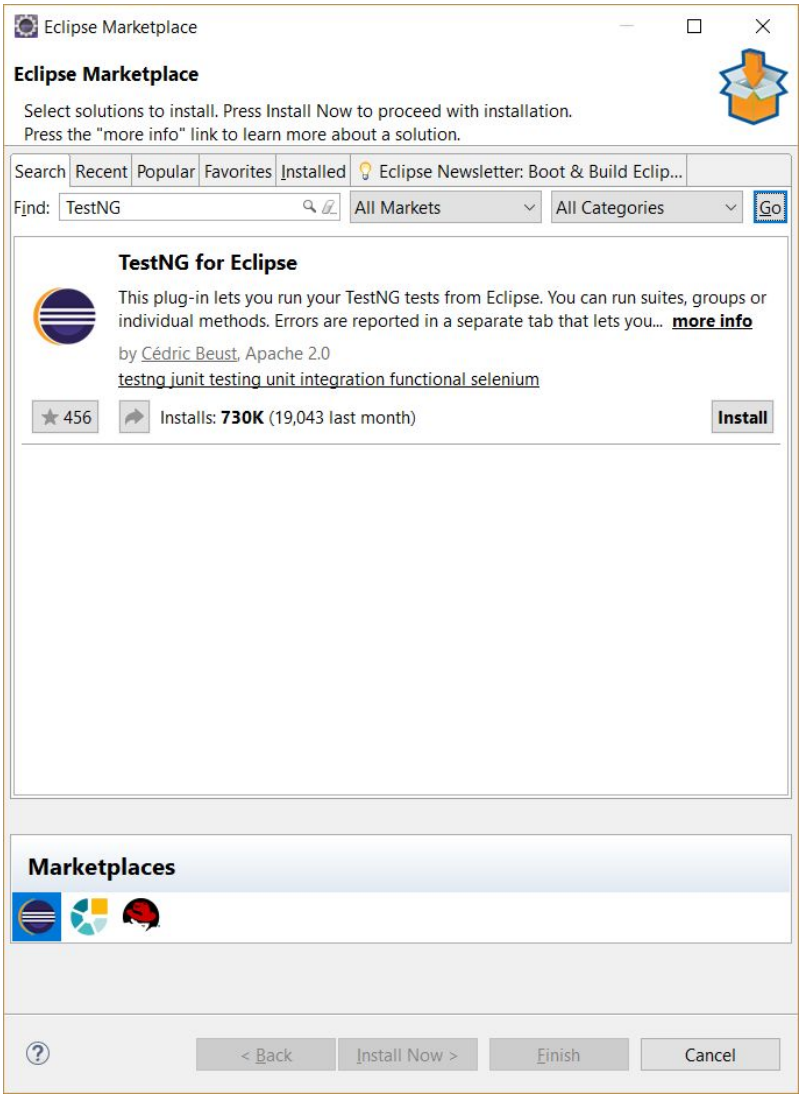
3- Obtener el código

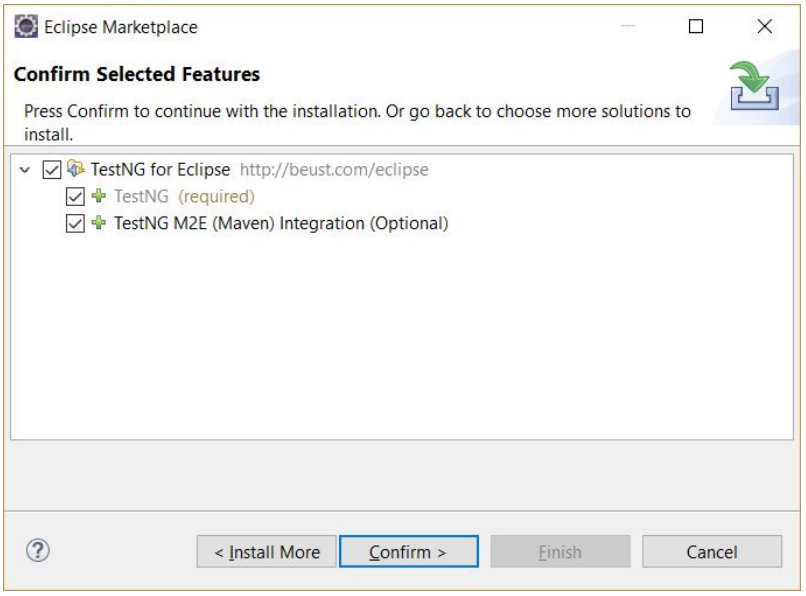
- Es recomendable tener instalado Firefox, pero se puede correr también con Chrome.
- Clonar el repositorio https://github.com/alexisfr/selenium-tests
- Cambiar al directorio: `cd selenium-tests`
- Ejecutar el comando: `mvn clean verify -Dbrowser=firefox -Dheadless=false` (Reemplazar por `-Dbrowser=chrome` si no dispone de Firefox)
- Esperar que finalice el proceso.

4- Configurar el Entorno de desarrollo

- Si utiliza IntelliJ, solo tiene que abrir el proyecto.
- Si utiliza Eclipse:

- Debe instalar el plugin de TestNG: En el menú Ayuda -> Eclipse Marketplace y buscar por TestNG



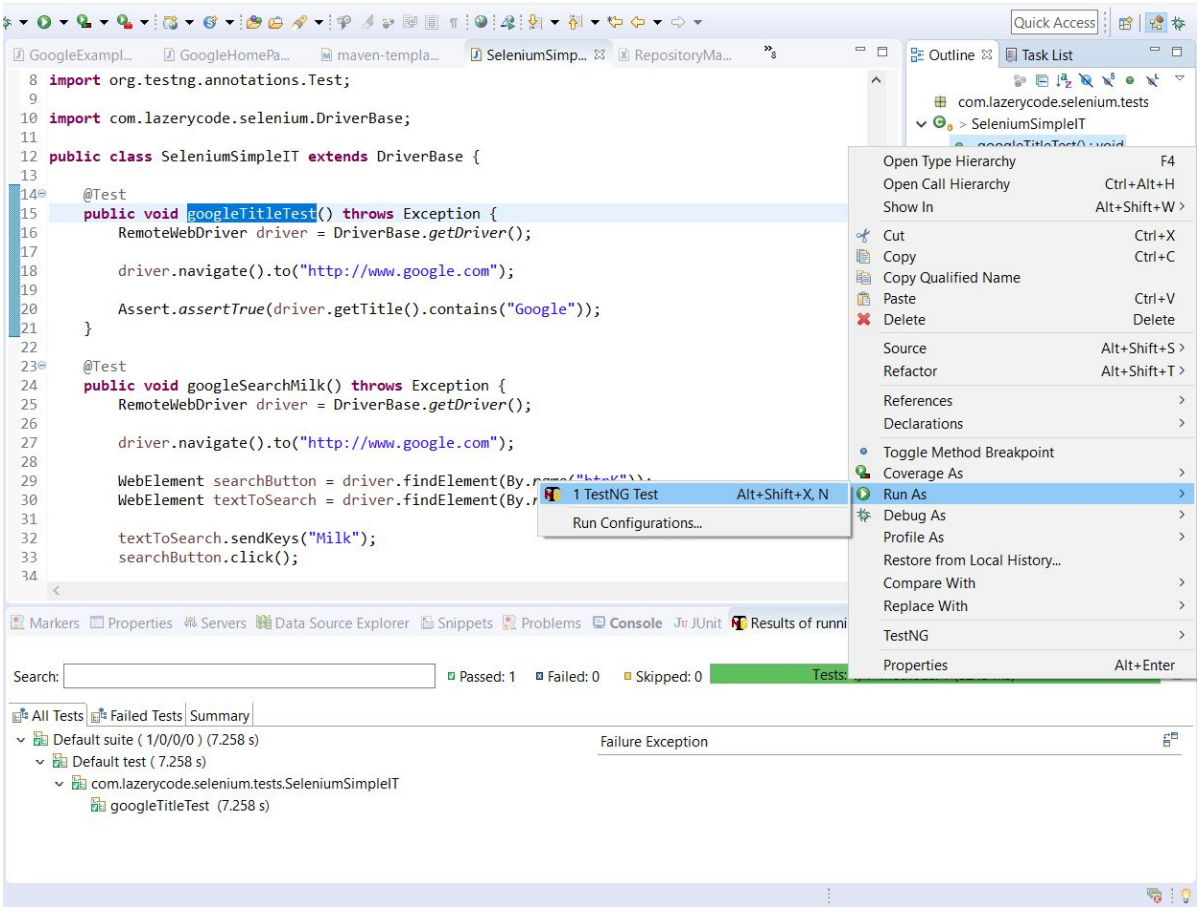


- El proceso puede tomar varios minutos.
- Importar el proyecto maven (Siguiendo los pasos explicados en el Trabajo Práctico 8)

- Modificar la propiedad webdriver.gecko.driver apuntando al binario correcto, para linux o osx (sin el .exe en estos últimos 2 casos). Algo similar a:

```
<webdriver.gecko.driver>${project.basedir}/src/test/resources/selenium_standalone_binaries/windows/marionette/64bit/geckodriver.exe</webdriver.gecko.driver>
```

- Ejecutar los test cases con la opción Correr como TestNG



5- Analizar las pruebas

- Abrir el archivo SeleniumSimpleIT.java
- Explicar los test cases que se encuentran en el archivo.
- Ejecutar ambos test cases. Ver los resultados.

6- Escribir una prueba

- Inspeccionar la página web www.google.com
- Utilizamos el navegador para encontrar ids de objetos y clases
- Cual es el name del botón Me siento con suerte?
- Escribir un test case que verifique si el logo está presente.
  - Una forma de lograrlo es utilizando wait más la función ExpectedConditions.presenceOfElementLocated

7- Analizar pruebas utilizando Page Object

- Abrir el Archivo GoogleExampleIT
- Explicamos los tests y la idea detrás de este tipo de patrón.